Università degli Studi di Parma
Dipartimento di Ingegneria e Architettura

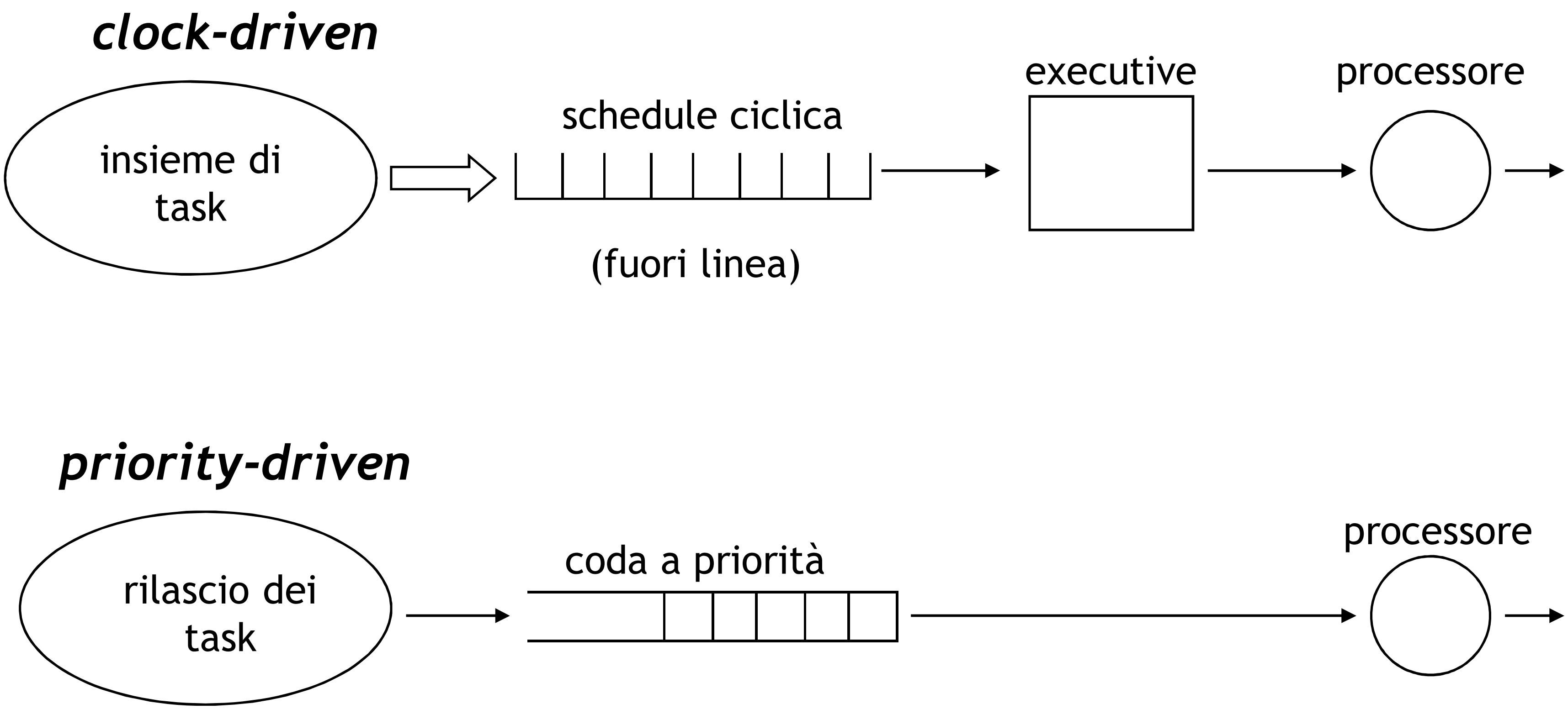
Sistemi operativi e in tempo reale – a.a. 2022/23

Scheduling di task periodici basato su priorità

prof. Stefano Caselli
stefano.caselli@unipr.it

Scheduling basato su priorità

- Priority-driven vs. clock-driven:



Scheduling di task periodici basato su priorità

- Ipotesi:
  - task indipendenti
  - task periodici
  - assenza di task aperiodici e sporadici
  - i task sono pronti non appena rilasciati e non si sospendono da soli
  - preemption consentita ovunque

- Le decisioni di scheduling sono prese in corrispondenza al rilascio ed al completamento dei job
  - overhead per cambio di contesto trascurabile
  - numero illimitato di livelli di priorità

Scheduling di task periodici basato su priorità

- Ipotesi di contesto:

- I risultati relativi ad insiemi di task *periodici* valgono anche considerando il periodo come *minimo tempo tra due rilasci* consecutivi di un task

- Il numero di task periodici ammessi nel sistema è fisso o regolato da un modulo di accettazione

- Facciamo riferimento a sistemi uniprocessore, o a sistemi multiprocessore in cui i task sono allocati *staticamente*

Algoritmi a priorità fissa e dinamica

- Uno scheduler priority-driven opera *on-line*: assegna priorità ai job quando vengono rilasciati e li inserisce nella coda dei job pronti in base alla loro priorità
- Algoritmi *a priorità fissa*: tutti i job di un task hanno la stessa priorità → la priorità del task è statica (es.: RM)
- Algoritmi *a priorità dinamica*: possono assegnare priorità diversa ai job di un task → la priorità *del task* è dinamica (es.: EDF)
- Normalmente *i singoli job* hanno comunque una priorità statica

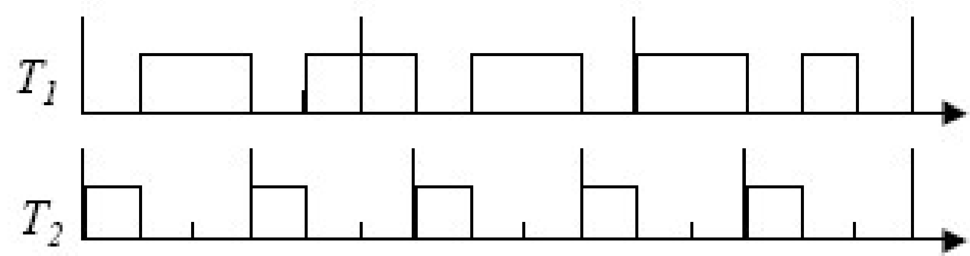
Classificazione degli algoritmi a priorità

- La priorità è attribuita ai job *quando vengono rilasciati* ed inseriti tra i job pronti
- Una volta assegnata, la priorità di un job rispetto *agli altri job pronti* non cambia

- Classificazione:
  - algoritmi a *priorità fissa o statica* (per task e job)
  - algoritmi a *priorità dinamica* a livello di task (a priorità fissa a livello di job)
  - algoritmi a priorità dinamica a livello di job (e quindi anche di task) –- poco frequenti

Algoritmo Rate Monotonic

- La priorità è attribuita ai task in base al loro *periodo*: minore è il periodo, maggiore è la priorità
- Esempio: T1=(5,3,5), T2=(3,1,3)



- T2 ha sempre priorità su T1, e ciò determina alcune situazioni di *preemption*
- *Priorità statica*: la priorità è proporzionale al *rate* del task (l'inverso del periodo)

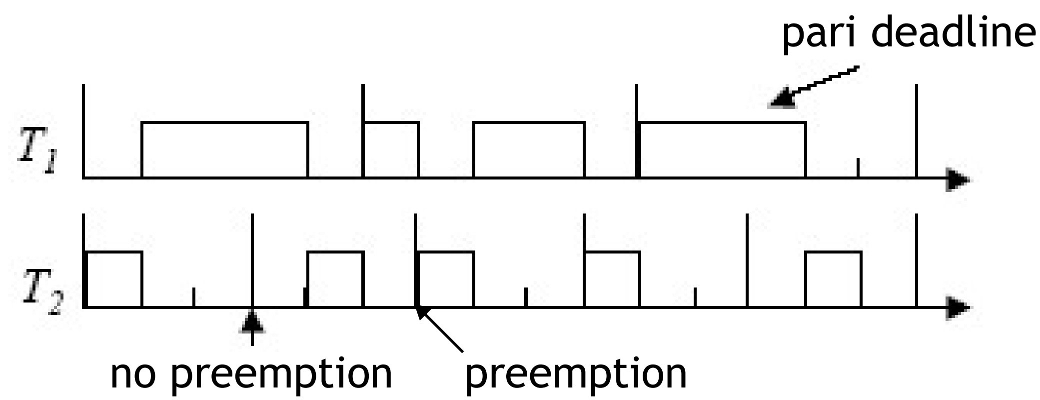
Algoritmo Rate Monotonic

- Ha senso?
- Se un task periodico ha un *rate elevato* (cioè un periodo più breve), tutti i sui job avranno scadenze ravvicinate
- → può essere utile attribuire ai job del task una priorità elevata, a fattor comune
- Stessa priorità per tutti i job → *priorità del task*
- Evita manipolazioni della priorità a tempo di esecuzione: *setting priorities? it's cheap but it ain't free*

Algoritmo Earliest Deadline First

- La priorità è attribuita ai job in base alla loro *deadline assoluta*: più prossima è la deadline assoluta, maggiore è la priorità
- Esempio:

  T1=(5,3,5), T2=(3,1,3)

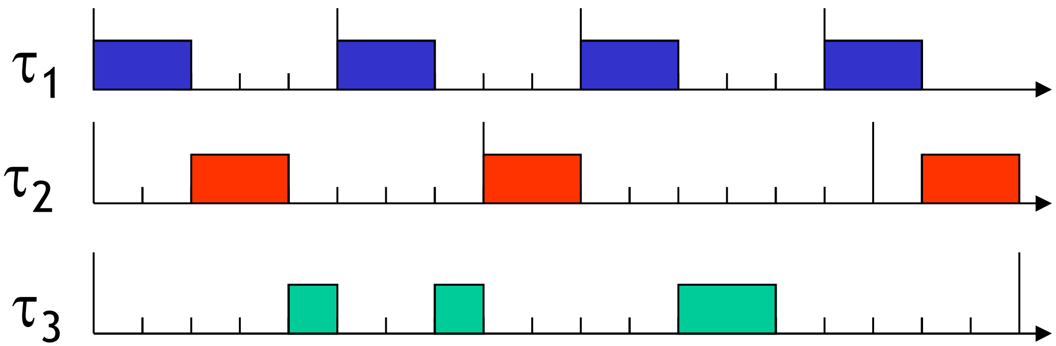


- La *preemption* si può ancora verificare, in base al diverso criterio di priorità
- *Priorità dinamica*: i job di Ti possono avere priorità relativa diversa rispetto ai job di un altro task periodico Tj

Algoritmo Earliest Deadline First

- La priorità EDF esprime direttamente l'urgenza del job
- Job diversi dello stesso task rilasciati nel tempo hanno priorità diversa
- Se Ti<Di, job diversi dello stesso task possono essere pronti per l'esecuzione, ma la politica EDF ne determina una diversa priorità e i job saranno eseguiti nell'ordine di rilascio
- Ad ogni rilascio di un job lo scheduler ne deve definire la priorità relativa rispetto agli altri job pronti --> per la realizzazione è utile una struttura *callback queue*

Algoritmo Rate Monotonic



- Negli algoritmi statici, i task a priorità maggiore sono "protetti"
- I job dei task a priorità non massima possono subire più revoche per periodo in funzione del numero e della frequenza dei task a priorità più alta

Algoritmi statici

- La priorità è stabilita in base a caratteristiche intrinseche *del task*, che valgono per tutti i suoi job: periodo, deadline relativa, "importanza"

- I principali sono *Rate Monotonic* (RM) e *Deadline Monotonic (DM)*, che ordina le priorità dei task in base alle deadline *relative*
- L'algoritmo RM (Liu e Layland, 1973) è *ottimo* tra gli algoritmi *statici* in cui la *deadline relativa coincida con il periodo*
- L'algoritmo DM (Leung e Whitehead, 1982) generalizza RM per *deadline arbitrarie*, ed è a sua volta *ottimo* in questa ipotesi più generale

Altri algoritmi statici

- Priorità legata all' "importanza" del task? Es.:
  - priorità basata su criticità funzionale
  - ordinamento alfabetico
  - etc.

- Tipicamente <u>non ottimi</u>
- Con alcuni strumenti può ancora essere possibile garantire uno o pochi task
- Non considerano i parametri temporali che determinano l'urgenza del task!
- → Limitiamo l'attenzione ad algoritmi priority-driven in cui la priorità è collegata a parametri temporali

Algoritmi dinamici

- La priorità è stabilita in base a caratteristiche *tempo varianti del task*: ad es. la deadline *assoluta* del job, l'istante di rilascio del job, etc.

- I principali sono *Earliest Deadline First (EDF)* e *Least Slack Time First (LST)*:
- EDF: considera la deadline assoluta, LST: lo slack time residuo

- FIFO e LIFO: considerano l'istante di rilascio del job; danno luogo a risultati scadenti perchè non si basano su un parametro correlato all'*urgenza* del job

Algoritmi dinamici: varianti

- *LST non stretto*: valuta gli slack time solo quando un job è rilasciato o completa
- *LST stretto*: deve valutare continuamente lo slack time del job in esecuzione; quando raggiunge lo slack di job in attesa, la schedulazione prosegue in modo *Round-Robin* per evitare instabilità

Considerazioni realizzative

- Dal punto di vista realizzativo, <u>gli algoritmi statici sono più semplici</u>:
  - la priorità viene attribuita staticamente
  - il job, quando rilasciato, viene direttamente inserito in coda al livello di priorità che gli compete

- Negli algoritmi dinamici occorre rivalutare la <u>priorità relativa ad ogni rilascio del job</u>
  - Ma ... (Spoiler)

Utilizzazione

- Ogni task periodico usa il processore per la frazione di tempo:

$$U_i = \frac{C_i}{T_i}$$

- L'*utilizzazione* totale del processore è: $$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- Il *(fattore di) utilizzazione U_p* è una misura del carico del processore
- Se i task richiedono $U_p > 1$ il processore è *sovraccarico* e l'insieme di task non può essere garantito

Utilizzazione schedulabile

- Definizione:

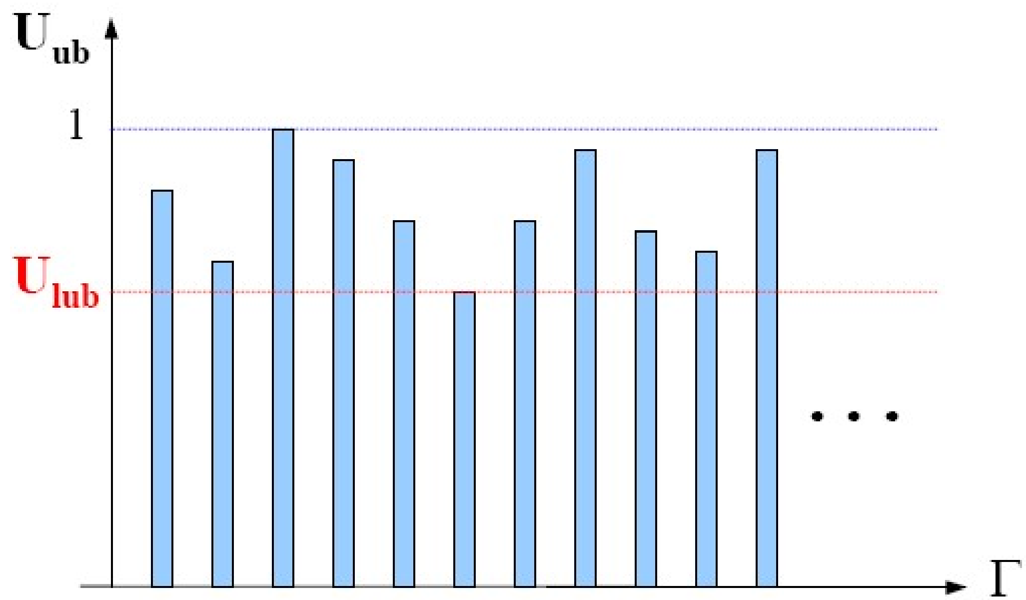
  valore del fattore di utilizzazione totale associato ad un algoritmo tale che ogni insieme di task periodici la cui utilizzazione totale è non superiore a tale valore è schedulabile dall'algoritmo
- ... maggiore l'utilizzazione schedulabile, migliore l'algoritmo
- E' sempre ≤ 1

- Denominata anche *Least Upper Bound* del fattore di utilizzazione, U_{lub}

Il Least Upper Bound

- Dato un algoritmo α, il valore massimo di U per cui i task sono schedulabili, U_{ub}, dipende dall'insieme di task Γ
- Il minimo di U_{ub} per cui qualunque insieme di task risulta schedulabile dall'algoritmo α è $U_{lub}(\alpha)$

- Se $U(\Gamma) \leq U_{lub}(\alpha)$, Γ è certamente schedulabile dall'algoritmo α



Utilizzazione schedulabile

- Un test di schedulabilità a basso costo:
- Dato un insieme di task periodici Γ ed un algoritmo di scheduling α, condizione *sufficiente* perchè Γ sia schedulabile da α:

$$U(\Gamma) \leq U_{lub}(\alpha)$$

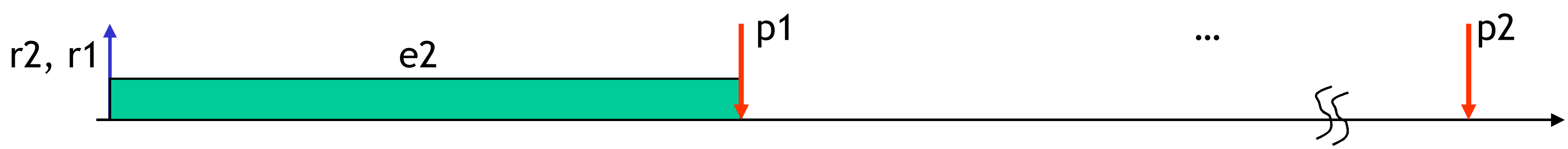
- Se il test non è soddisfatto ma $U(\Gamma) \leq 1$, *Γ può essere schedulabile oppure no* da α

Utilizzazione schedulabile per FIFO

- Teorema: $U_{lub}(FIFO)=0$

- Dim: Dato un qualunque livello di utilizzazione $\varepsilon>0$, è possibile costruire un task set con utilizzazione ε ma *non schedulabile* in modo FIFO
- Es.: $\tau 1=(p1,e1)=(p, \varepsilon\ p/2)$, $\tau 2=(p2,e2)=(2\ p/\varepsilon, p)$

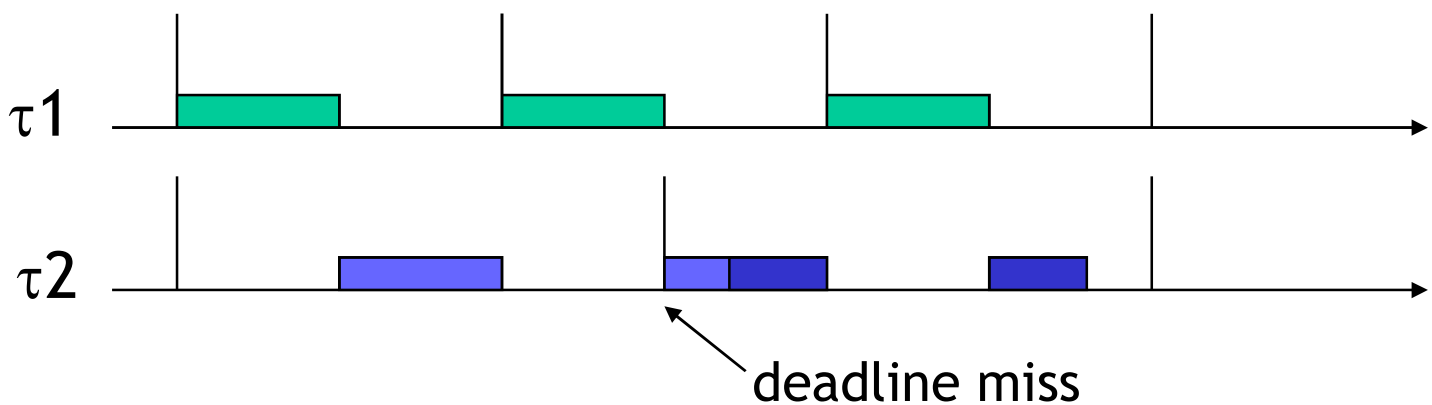
  $U1=(\varepsilon\ p/2)/p=\varepsilon/2$, $U2=p/(2p/\varepsilon)=\varepsilon/2 \rightarrow U=\varepsilon$
- se $\tau 1$ arriva appena dopo $\tau 2$, $\tau 1$ manca la deadline



Utilizzazione schedulabile per RM

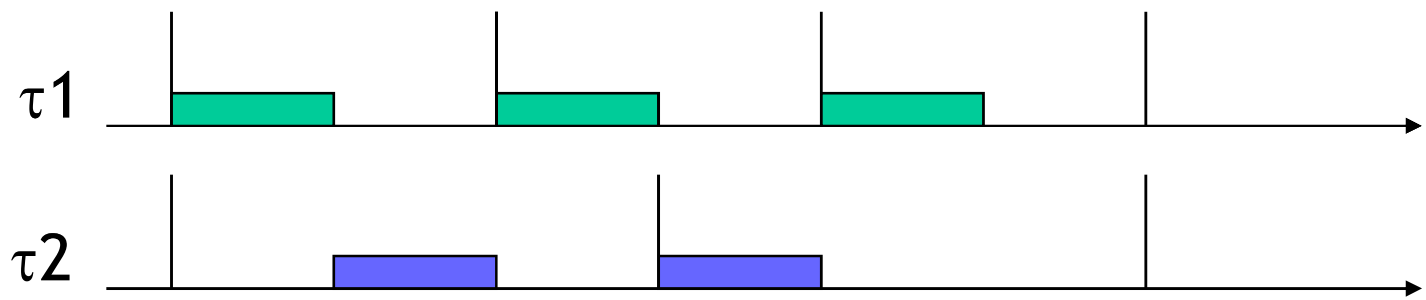
- Teorema: $U_{lub}(RM) < 1$

- Esempio: $\tau 1=(6,3)$, $\tau 2=(9,4)$
- $U=3/6+4/9= 0.944 < 1$



Utilizzazione schedulabile per RM

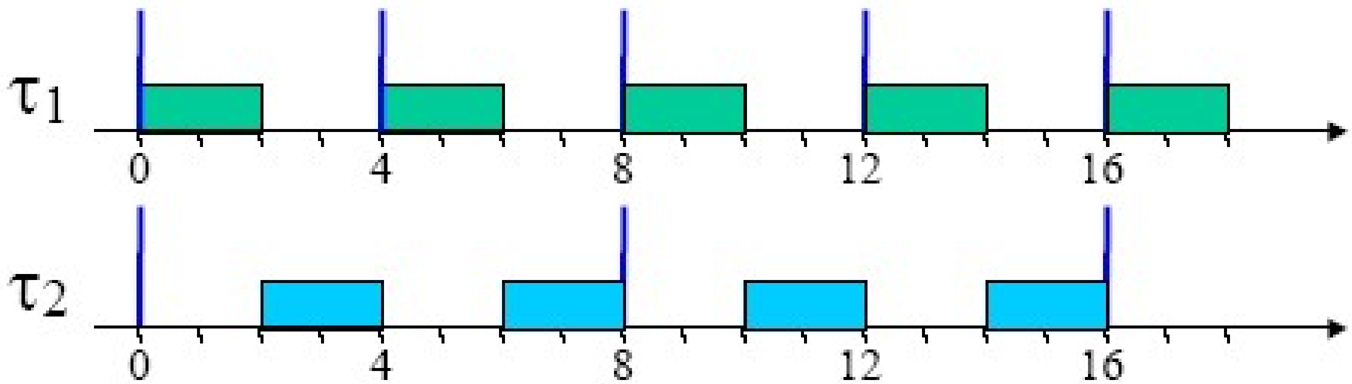
- Esempio: $\tau 1=(6,3)$, $\tau 2=(9,3)$
- $U=3/6+3/9=0.833$



- Nota: se *e1* o *e2* vengono aumentati, $\tau 2$ non rispetterà la deadline
- Situazione di *piena utilizzazione* del processore

Utilizzazione schedulabile per RM

- Esempio: $\tau 1=(4,2)$, $\tau 2=(8,4)$
- U=1



- ➔ $U_{ub}(\Gamma)$ dipende non solo dal numero di task, ma anche dai parametri temporali dei task in Γ. U_{lub}?

Utilizzazione schedulabile per RM

- Teorema (Liu e Layland, 1973)
  Least Upper Bound per scheduling RM:

$$U_{lub}(n)= n(2^{1/n} - 1)$$

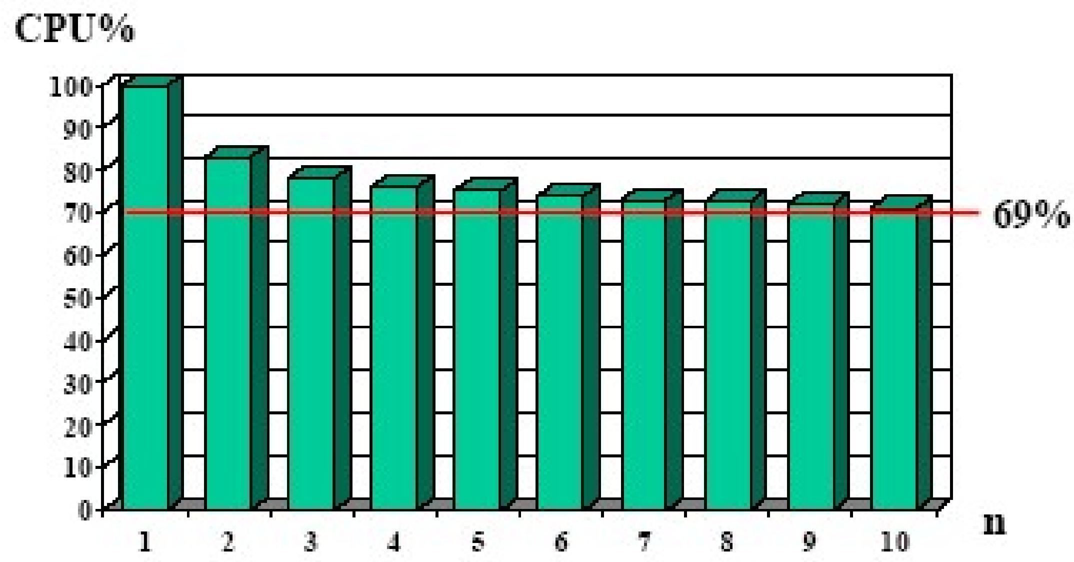
- Il limite inferiore di U_{lub}(RM) *dipende solo dal numero di task*
- Per $n \to \infty$ $\quad U_{lub}(RM) \to \mathit{ln}\, 2 \cong 0.693$

Utilizzazione schedulabile per RM

$U(n)= n(2^{1/n} - 1)$

U(1)=1.0 U(2)=0.828 U(3)=0.779 U(4)=0.756

U(5)=0.743 U(6)=0.734 U(7)=0.728 U(8)=0.724



Test di Liu e Layland per garanzia RM

- Calcoliamo l'*utilizzazione richiesta* del processore come:

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- L'insieme di task è *garantito se* (solo sufficiente):

$$U_p \leq n(2^{1/n} - 1)$$

- E' un test poco costoso, O(n)
- Se $n(2^{1/n} - 1) < U_p \leq 1$, l'insieme di task può essere *schedulabile* oppure no

Ipotesi dello scheduling RM

- e_i costante per ogni job di T_i
- p_i costante per ogni job di T_i
- $D_i=p_i$ per ogni task T_i
- Task indipendenti: non ci sono vincoli di precedenza e vincoli su risorse

Ipotesi dello scheduling RM (Liu e Layland, 1973)

1. Le *richieste* di tutti i task con hard deadline sono *periodiche*, con intervalli costanti tra due richieste successive.
2. Per ciascun task il solo vincolo temporale è la sua *completa esecuzione* prima della successiva richiesta di attivazione.
3. I *task* sono *indipendenti*: le richieste di attivazione di un task non dipendono dall'inizio o dal completamento dell'attivazione di altri task.
4. La *durata dell'esecuzione* di ciascun task è *costante* (a meno delle eventuali interruzioni del task).
5. Gli eventuali *task non periodici* sono *non critici* o sono comunque esclusi dall'analisi.

Ottimalità dello scheduling RM

- Teorema: Nelle ipotesi sopracitate, *RM è ottimo tra tutti gli algoritmi a priorità fissa*.

- Se esiste un assegnamento a priorità fissa che genera una schedule fattibile per Γ, anche l'assegnamento RM è fattibile per Γ
- ➔ se Γ non è schedulabile da RM, allora non è schedulabile da alcun assegnamento a priorità fissa, <u>nelle ipotesi date</u>

Istante critico

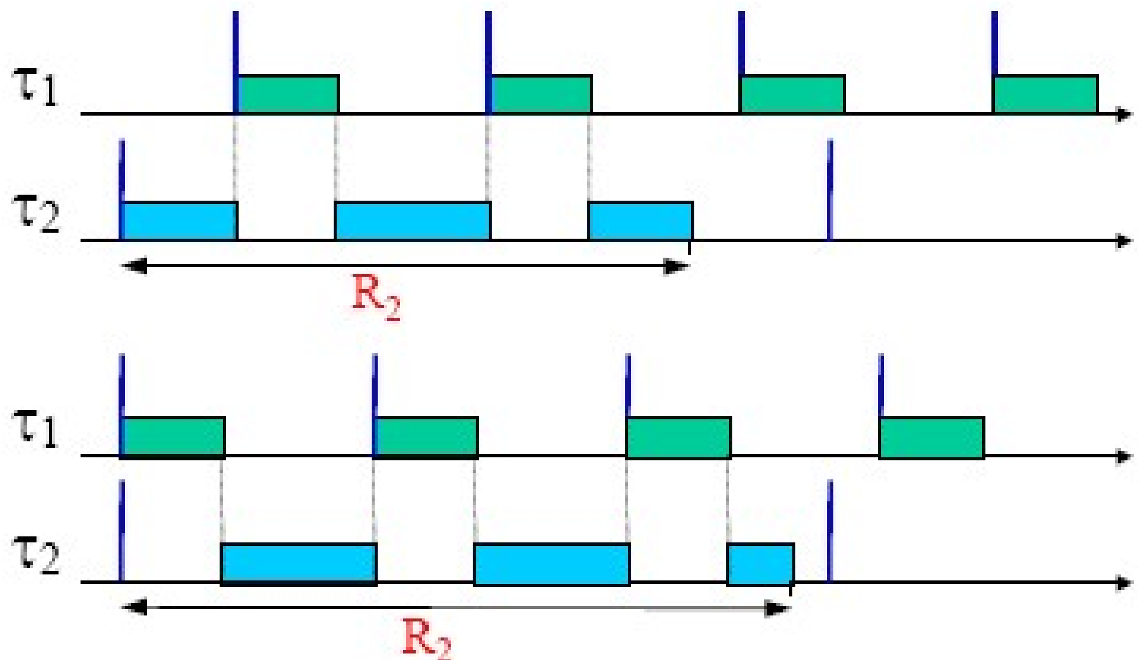
- *Istante critico* di un task: è l'istante in cui una richiesta di attivazione del task determina il massimo tempo di risposta
- *Intervallo critico* di un task: è il tempo che intercorre tra un istante critico ed il termine della risposta alla richiesta corrispondente

- Fissato l'insieme di task Γ, se un task è schedulabile nel suo intervallo critico (→ rispetta la sua prima deadline) lo è certamente per qualsiasi relazione di fase tra i task
- Possibile tecnica di analisi: esame degli intervalli critici per tutti i task

Istante critico

- Teorema: In un algoritmo con assegnamento statico delle priorità, per ogni task τ_i il massimo *tempo di risposta* R_i si verifica quando esso viene rilasciato simultaneamente a tutti i task a priorità superiore

- Es. Pri(τ1)>Pri(τ2):

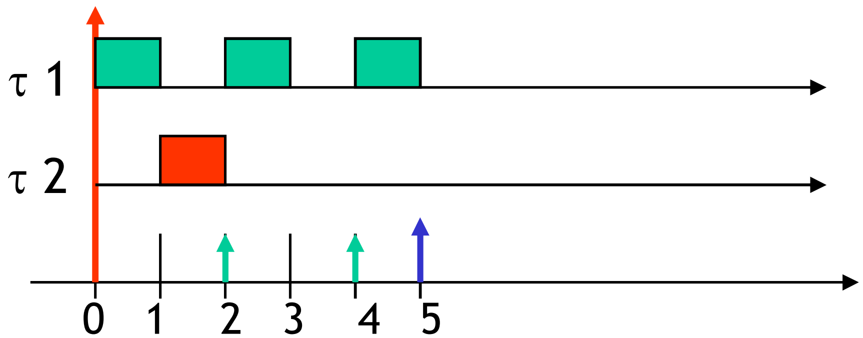


Intervallo critico e analisi di schedulabilità

- Un metodo per studiare la schedulabilità di un insieme di task:
- Analizzare la schedulabilità dei task a priorità inferiore *a partire dall'istante critico*
- Vale per tutti gli algoritmi di scheduling *statici*
  - Anche non RM e non DM, anche con priorità assegnate in modo arbitrario

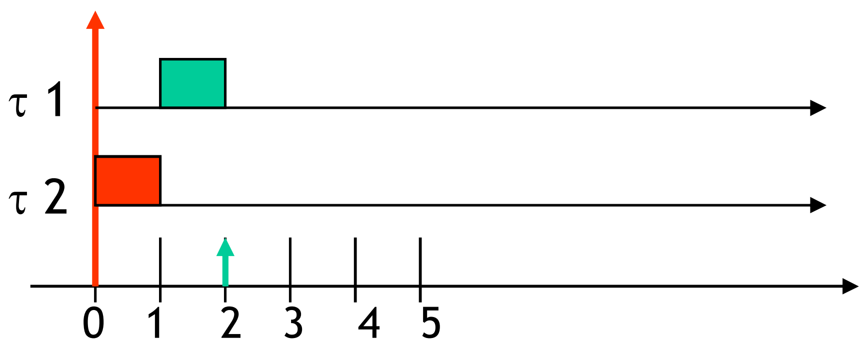
Analisi degli intervalli critici

- Esempio $\tau 1=(2,1)$, $\tau 2=(5,1)$
- Se Pri(τ1)>Pri(τ2) → esame dell'intervallo critico di τ2:



- Lo scheduling é fattibile e rimane tale anche incrementando C2 fino a C2=2

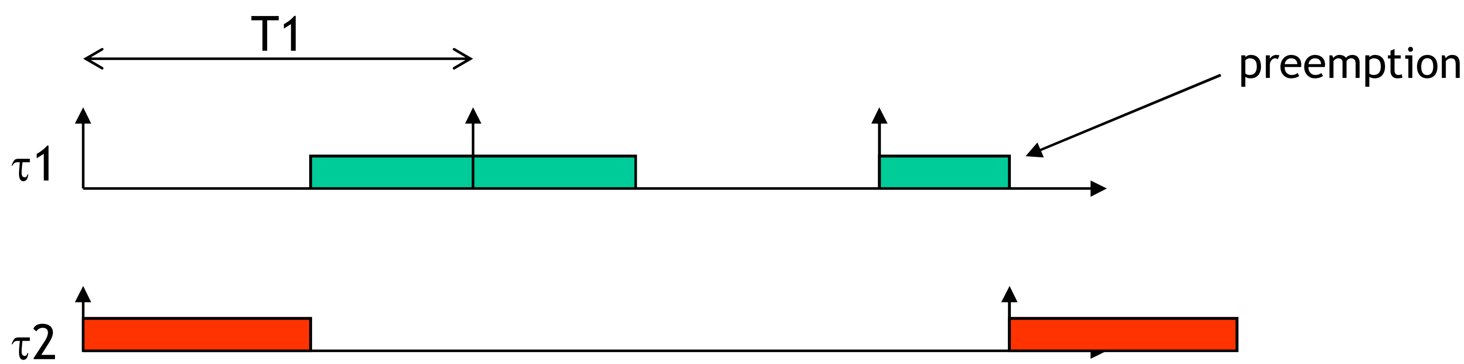
- Se Pri(τ2)>Pri(τ1) → esame dell'intervallo critico di τ1:



- Lo scheduling é fattibile, ma né C1 né C2 possono essere aumentati
- Il processore è *pienamente utilizzato*

Dimostrazione dell'ottimalità di RM

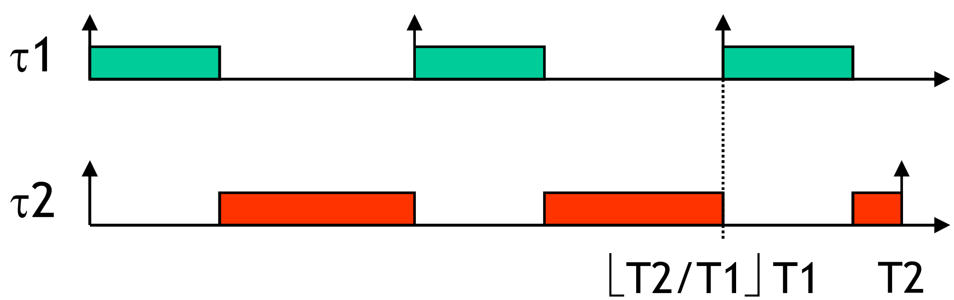
- Consideriamo due task $\tau 1$ e $\tau 2$ con periodi $T1<T2$
- Assegnamo la priorità in modo *non RM*: $Pri(\tau 2)>Pri(\tau 1)$
- La schedule è fattibile se: $C1+C2 \leq T1$ (1)



- Il processore è pienamente utilizzato (caso peggiore)

Dimostrazione dell'ottimalità di RM

- Se assegnamo la priorità in modo RM: $Pri(\tau 1) > Pri(\tau 2)$
- Si possono verificare due situazioni:
  - (a) Tutte le richieste di $\tau 1$ entro l'intervallo critico di $\tau 2$ sono completate
  - (b) Durante l'esecuzione dell'ultima richiesta di $\tau 1$ si verifica un nuovo rilascio di $\tau 2$

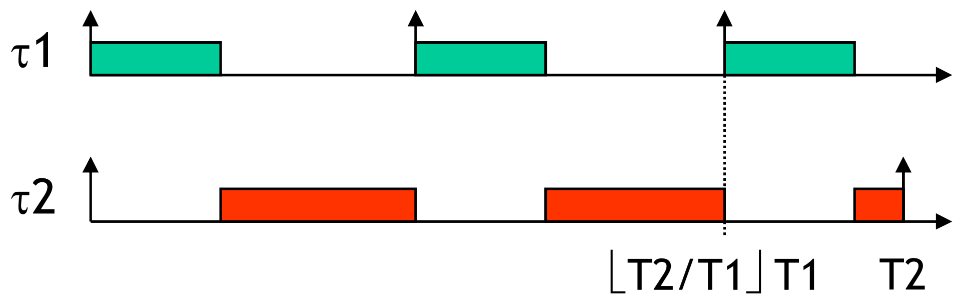


- Caso (a) (def):

$$C1 \le T2 - \lfloor T2/T1 \rfloor T1$$

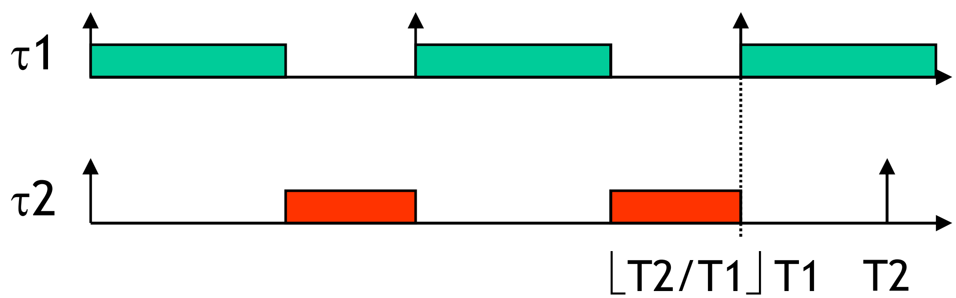
Dimostrazione dell'ottimalità di RM

- Caso a):



$$C1 \leq T2 - \lfloor T2/T1 \rfloor T1$$

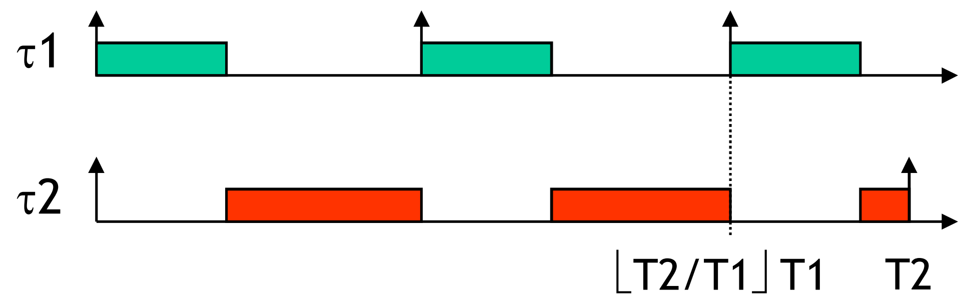
- Caso b):



$$C1 \geq T2 - \lfloor T2/T1 \rfloor T1$$

Dimostrazione dell'ottimalità di RM

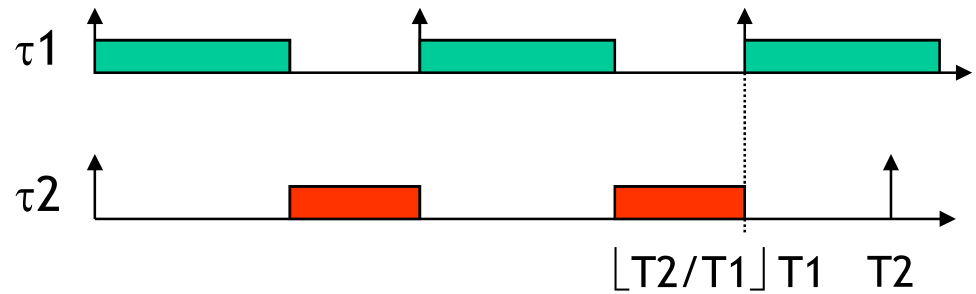
- *Caso a):*
  $C1 \leq T2 - \lfloor T2/T1 \rfloor T1$



- La schedule è fattibile se:
  $(\lfloor T2/T1 \rfloor + 1)C1 + C2 \leq T2$ (2)
- Dimostriamo che se vale la (1), vale anche la (2)
- Moltiplicando la (1) si ha:
  $\lfloor T2/T1 \rfloor C1 + \lfloor T2/T1 \rfloor C2 \leq \lfloor T2/T1 \rfloor T1$
- Essendo $\lfloor T2/T1 \rfloor \geq 1$:
  $\lfloor T2/T1 \rfloor C1 + C2 \leq \lfloor T2/T1 \rfloor C1 + \lfloor T2/T1 \rfloor C2 \leq \lfloor T2/T1 \rfloor T1$
- Sommando C1 ad ogni membro e poichè $C1 \leq T2 - \lfloor T2/T1 \rfloor T1$:
  $(\lfloor T2/T1 \rfloor + 1)\ C1 + C2 \leq \lfloor T2/T1 \rfloor T1 + C1 \leq T2$

Dimostrazione dell'ottimalità di RM

- *Caso (b):*
  $C1 \geq T2 - \lfloor T2/T1 \rfloor T1$



- La schedule è fattibile se:
  $$\lfloor T2/T1 \rfloor C1 + C2 \leq \lfloor T2/T1 \rfloor T1 \qquad (3)$$
- Dimostriamo che se vale la (1), vale anche la (3)
- Moltiplicando la (1) si ha:
  $$\lfloor T2/T1 \rfloor C1 + \lfloor T2/T1 \rfloor C2 \leq \lfloor T2/T1 \rfloor T1$$
- Essendo $\lfloor T2/T1 \rfloor \geq 1$:
  $$\lfloor T2/T1 \rfloor C1 + C2 \leq \lfloor T2/T1 \rfloor C1 + \lfloor T2/T1 \rfloor C2 \leq \lfloor T2/T1 \rfloor T1$$

Dimostrazione dell'ottimalità di RM

- Abbiamo dimostrato che se vale (1), vale (2) nel caso (a) e vale (3) nel caso (b) → anche la schedule RM è fattibile

- Estendendo la discussione ad n task e riordinando le priorità tra coppie di task, si dimostra che l'algoritmo RM è ottimo

Applicazione della teoria RM

- Esempio:

  $\tau 1=(100, 20)$ $U1 = 0.2$

  $\tau 2=(150, 40)$ $U2 = 0.267$

  $\tau 3=(350, 100)$ $U3 = 0.286$

- Utilizzazione complessiva richiesta dai 3 task:

  $$U = 0.753 \leq U^*(3) = 0.779$$

- I task sono garantiti e schedulabili *se* lo scheduling é RM, cioè $Pri(\tau 1) > Pri(\tau 2) > Pri(\tau 3)$

Applicazione della teoria RM

- Se $U > U^*(m)$ occorre verificare gli intervalli critici (notazione: $U^*(m) = U_{lub}(RM)$ per m task)

- Teorema dell'*intervallo critico*: se ogni task rispetta la propria prima deadline quando tutti i task richiedono l'attivazione allo stesso istante, allora tutte le deadline verranno comunque rispettate per qualunque combinazione degli istanti di richiesta

- Modifica dell'esempio precedente con C1=40:

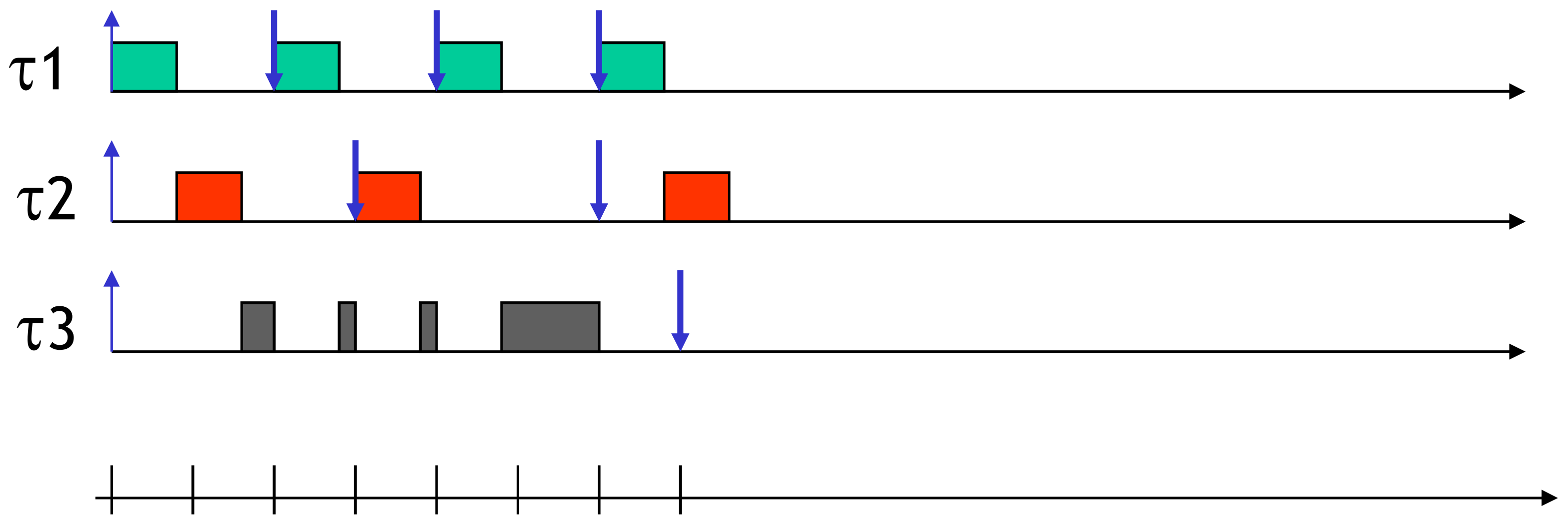
$$U = 0.953 > U^*(3) = 0.779$$

Applicazione della teoria RM

- $\tau 1=(100, 40)$ $\tau 2=(150, 40)$ $\tau 3=(350, 100)$ U = 0.953
- $\tau 1$ e $\tau 2$ hanno utilizzazione pari a 0.667 ≤ U*(2), pertanto *comunque rispettano le proprie deadline*
- Per $\tau 3$ si applica il Teorema dell'intervallo critico, verificando se completa la sua esecuzione entro l'intervallo T3
- La schedulazione statica *protegge i task a priorità più elevata*; le analisi più onerose sono necessarie solo per i task a priorità più bassa

Applicazione della teoria RM

- In questo caso si verifica che anche $\tau 3$ rispetta la sua deadline
- $\tau 1=(100, 40)$ $\tau 2=(150, 40)$ $\tau 3=(350, 100)$



Considerazioni pratiche

- Utilizzazione schedulabile di RM per Liu e Layland:

  $$U_{lub}(RM) \leq n(2^{1/n} - 1)$$

- Per $n \to \infty$ $\quad U_{lub}(RM) \to \ln 2 \cong 0.693$
- Se $n(2^{1/n} - 1) < U_p \leq 1$, l'insieme di task può essere schedulabile oppure no
  - task set *schedulabile* secondo il bound LL? → forse
  - task set *garantito* dal bound LL? → no

- Il bound $U_{lub}(RM)$ di Liu e Layland è un bound *pessimistico*
- Studi su insiemi di task periodici generati casualmente hanno riscontrato un valore probabile del bound di 0.88

Esercizi su scheduling RM con bound Liu-Layland

- Task set in diapositiva n. 41:

  $\tau 1=(100, 20)$ $\tau 2=(150, 40)$ $\tau 3=(350, 100)$ U = 0.753
- I task schedulati <u>in modo RM</u> sono *garantiti* dal bound di Liu-Layland (bound LL) e il task set è *schedulabile*

- <u>Esercizio</u>: Analizzare i seguenti assegnamenti di priorità statica (non RM):
  - pri(τ2)>pri(τ1)>pri(τ3)
  - pri(τ2)>pri(τ3)>pri(τ1)
  - pri(τ3)>pri(τ2)>pri(τ1)
  - pri(τ3)>pri(τ1)>pri(τ2)
  - pri(τ1)>pri(τ3)>pri(τ2)

Esercizi su scheduling RM con bound Liu-Layland

- Cosa accade con altri assegnamenti statici di priorità, diversi da RM? Quali tecniche di analisi possiamo utilizzare?

- Per ogni assegnamento di priorità elencato, utilizzando le tecniche e i teoremi visti a lezione, specificare se:
  - ciascun task è individualmente garantito (e perché) -> risposta (si|no)
  - il task set è schedulabile -> risposta (sì|no|forse)